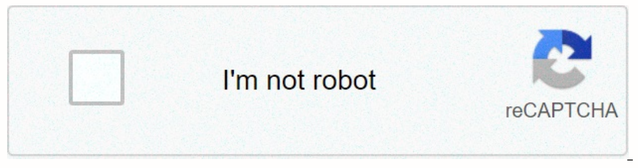

Continue

Protobuf java api

More than 3 years have passed since the last update. Protocol buffers communicate in binary format without communicating in text formats such as Json, such as API communications from proto files (predefined files), write files for serialization and deseriaization to the server and client side, and use them to send and receive data. The definition of the proto file this time, I want to create an API that returns a list of provinces. First, define the master file as follows: province grammar.proto = proto3; options java_package = com.takasumba.gouda.protos; option java_outer_classname = province go_package; (2; // String name) romaji = 3; // Reading (Romaji) } Text GetPrefecturesResponse { Repeat perfect province = 1; } The actual proto code is here, write the java file from Proto from the pro file you previously defined, write for the serialization and desery. I'll write it off using a wire made by a table here. and put the proto in the folder you define. After that, you can write java files with the following command: Java-jar wire compiler - VERSION-jar-with-dependncies.jar \--proto_path =your /proto/path\ - java_out = your /output/path, I think you can write such a file. Add the library needed to the gradient for API communication in the Proto// retrofit compilation com.squareup.retrofit2:retrofit:$retrofitVersion compiler.com.squareup.retrofit2:adapter-rx java2:$retrofitVersion Compile com.squareup.retrofit2:converter-wire:$retrofitVersion compiled com.squareup.okhttp3:gging-interkeeper:$okhttpVersion compiled com.squareup.okhttp3:okhttp:$ client assignment of okhttpVersion adapted this time I will write using personal kotlin val OKhttp:$ client schedule of okhttp.Client: OKHttpClient = OKHttpClient Builder() .addNetworkInterceptor (httpLoggingInterkeeper()setLevel (httploggingIntercever.Level.HEADERS)).connectTimeout(15, TimeUnit.SECONDS) .readTimeout(15, TimeUnit.SECONDS) .writeTimeout(15, TimeUnit.SECONDS) .build() Personal valtoRetrofit: Retrofit Builder() .baseUrl (destination), .client (okHttpClient), .addConverterFactory (WireConverterFactory.create()) .addCallAdapter Factory(RxJaxCallAdapterFactory.create()) .build(). Determine the type of response is the type of ptoro generated (in this case, GetPrefecturesRespone) interface service { @GET (province) enjoy getProtoProtoProrefectures(): Single } Hit Fun API<GetPrefectuResponse> GetProtoProtoFrefectures(): > Single { <> <Prefecture>Back protoRetrofit .create (Service:Class.java) .getProto Prefecture() .map (GetPrefecturesResponse::p refectures) .map { val:<Prefecture> ArrayList = ArrayList() it.mapTo (province) { value -> Province() Use { id = value.id = value.name romaji = value.romaji} } Province.toList() } }//Hitting APIClient.getProtoProfefectures() .doOnSubscribe({dispos) cans.add(it) .subscribeOn (Schedulers.io().observeon(AndroidSched). {ulers.mainThread()) .subscribe ({ toString()), { Timber.d(Error: + it.message) }) github: CyberAgent operates abemaTV, an Internet television station and has the highest share in Japan, with the vision that creating a company that represents the 21 century, we continue to expand our business while transforming the Internet industry by transforming new businesses and get more from Qiita? We will do it</Prefecture></Prefecture></GetPrefecturesResponse> We will do it</Prefecture></Prefecture></GetPrefecturesResponse> Articles that match youBy user trackers and tags, you can track information about the technical fields you invest as wholeyou can read useful information later effectively By. 1 year ago Articles you want, you can find immediately, you can register the main protocol buffer library Login. Protocol buffers are a structured but expandable way of crashing data in an efficient but expandable format. Central (61)Redhat GA (3)ICM (2)VersionRepositoryUsagesDate4.0.x4.0.0-rc-2Central4Jul, 20204.0.0-rc-1Central1Jul, 20203.14.x3.14.0Central17Nov, 20203.14.0-rc-3Central1Nov, 20203.14.0-rc-2Central1Nov, 20203.14.0-rc-1Central1Nov, 20203.13.x3.13.0Central439Aug, 20203.13.0-rcCentral1, 20203.12Aug.x3.12.4Cent281, Jul 20203.12.2Central312May, 20203.12.1Central35May, 20203.12.0Central55May, 20203.12.0-rc-2Central1May, 20203 .12.0-rc-1Central1May, 20203.11.x3.11.4Central425Feb, 20203.11.3Central201Feb, 20203.11.1Central290, ธ.ค. 20193.11.0Central81Nov, 20193.11.0-rc-2Central1Nov, 20193.11.0-rc-1Central1Nov , 20193.10.x3.10.0Central239กันยายน, 20193.10.0-rc-1Central2Sep, 20193.9. x3.9.2Central14Sep, 20193.9.1Central91Aug, 20193.9.0Central62Jul, 20193.9.0-rc-1Central1Jun, 20193.8.x3.8.0Central131May, 20193.8.0-rc-1Central1May, 20193.7.x3.7.1Central164Apr, 20193.7.0Central138Mar, 20193.7.0-rc1Central3Jan, 20193.6.x3.6.1Central362Jul, 20183.6.0Central112มิถุนายน, 20183.5.x3.5.1Central317Dec, 20173.5.0Central54Nov, 20173.4. x3.4.0Central199Aug, 20173.3.x3.3.1Central72May, 20173.3.0Central74May, 20173.2.x3.2.0Central135Jan, 20173.2.0rc2Central1Jan, 20173.2.0-rc.1Central1Jan, 20173.1.x3.1.0Central129Sep, 20163.0.x3.0.2Central62กันยายน, 20163.0.0Central146Jul, 20163.0.0-เบต้า-4Central1 2 Jul 2013.0.0-beta-3Central43May, 20163.0-beta-2Central81Jan, 20163.0.0- Beta-1Central30Aug, 20153.0-alpha-3.1Central3Jun, 20153.0-alpha-3Central8May, 20153.0-alpha-2CentralMar8, 20152.6.x2.6.1Central318October, 20142.6.0Central55Sep, 20142.5.x2.5.0Central680Mar, 20132.4.x2.4.1Central218, 20112.4.0aCentral37Mar, 20112.3.x2.3.0Cent 59February, 20102.2.x2.0Central26Oct, 20092.1.x2.1.0Central6Oct, 20092.0. x2.0.3Central6Feb, 20102.0.1Central 0 Oct. 2009 Hi, I'm Kado. Who is responsible for Boltsenken? If you're designing the REST API, do you sometimes have trouble mapping resources and features, or do you think it would be tricky to define the same format on your server and client ID? Today, I recommend the gRPC framework used in BoltzEngine gRPC, a new RPC framework created by Google, data that encodes the http/2 protocol buffer repeatedly in two roles: a gRPC server that provides an RPC method, and a gRPC client that calls the development method using gRPC determines how and if a text type (object) in a protocol buffer (.proto) is not dependent on the programming language and generates code for programming languages according to the above definition file. Series) Go Ruby C# Node.js Android Java Objective-C PHP gRPC Servers and gRPC clients can be used in other languages. For example, if the server side writes in Go. Python format of exchanged messages in RPC is automatically generated, so the cost of operation and maintenance is no different from using the same language. Prepare the protocol buffer compiler to compile the protocol buffer into another language. Prepare the protocol buffer compiler to compile the protocol buffer into another language. In the case of: # install protoc (one of the following) $ brew Install protobub #Homebrew if you use $nix-env -i protobuf-3.0.0 #Nixpkgs # Install runtime protocol buffer for Go $go get -u github.com/golang/protobuf/protoc-gen-go protoc, be sure to install 3.0 or higher 2.x, do not compile the proto file for gRPC to write the protocol development file definition by using gRPC as described above and the protocol protocol message to prepare the protocol protocol. Syntax = proto3; The first line syntax = proto3 specifies the version of the protocol buffer. Now it is a good idea to identify the proto3 blog service assigned services that gRPC provides. In this blog, if you write how the service needs them, they are treated as a method in gRPC, for example, RPC Send (message) will return (results); The parenthesis behind the method name displays the type of request message. The parenthesis behind the return is the type of reply. For each request and response method, gRPC can be selected as a simple text or stream, and the combination is divided into four categories: request and reply is simple (Simple RPC), the form of one response per request. This method does not have a definition but a stream. rpc MethodName (message) returns (reply); The format in which multiple responses are sent from the server in the wake of requests from RPC client-side streaming servers per request. The server prompts the client to end the response. This category has a stream response side. rpc MethodName (message) Returns (reply to a stream); In contrast to the above, it is the format in which the customer sends multiple requests and sends a response from the server once after all submissions have been sent. The side of the request becomes streamed and does not respond to rpc MethodName (streaming message). Return (reply); Multiple requests and multiple responses (rpc, bidirectional streaming), both the client and the server send and receive data multiple times. If you add a stream to both the request and the reply, here is it rpc MethodName will return (stream reply); Message type definition The type of message exchanged by the gRPC method must be defined in the protocol buffer with text { string from =1; Tags cannot be duplicated, and one-time active field tags must not be changed even if the field is deleted. Field types support common programming language types, such as bool, int32, uint64, double, string, and so on. In the following, I will introduce a slightly unusual category. If you attach repeated replies before the type The field is treated as an array or URL list of duplicate strings = 1; Type enum { INFO = 0; Warning = 1; Error = 2; Error } Text log { type = type = 1; } Map. Reply message { one of the text { plain string = 1; Before compiling the master file that you created in the language in which you actually write code, you will use Go here, but you can compile it with similar commands in other languages. mbox.proto --go_out=Plugin=grpc: go_out The compilation option must be specified successfully, but in this case, the code for the gRPC method must not be created. Specifies where to create the file, go_out the options available for the game are written in the official data parameter. If successful, a file called mbox.pb.go will be created, mbox.pb.go contains the necessary code for both the server and the client. Take a quick look at the code created before you start using it. Main import svr.go package (fmt log net pb ./proto golang.org/x/net/context google.golang.org/grpc) service type struct{} func (*Service) send (context ctx context, context, tonic powder *pb text) (*pb. Println (To:, HEROthing up) Body:= GetBody's HERO POWDER() If body == nil { returns nil, fmt error (missing) } Switch v:= Body (type) { case *pb. Message_Text: println fmt(text:, v.Text) typed *pb Message_Html: println fmt (HTML:, v.Html) } } returns && pb results{}, nil } func (*service) received (*pb Mail_ReceiveServer. Message{ and pb. Message{ from: from: from: from 1, to: [string {to 1, to 2}, subject: subject: body: &pb. Message_Text{ text: plain text, }, }, &pb{ message{ from: from: from 2, to: [string to to4}, Subject: Story2, Body text: &pb. Message_Html { Html: <div>Accompanying text</div>, }, }, } for _, r := reprіes range { if err := stream, send(r); err!= nil return { nil } Returns nil } func main() { l, err := net Listen(tcp, :13009) if err != nil { log Fatalln(err) } s:= grpc. Compiling a proto file is the same as on the server, so skip the file to the main import package cli.go (fmt io save pb ./proto golang.org/x/net/context google.golang.org/grpc) func main() {conn, err := grpc dial (localhost:13009, grpc. with insecure()) if err!= nil { save Fatalln (Dial:, err) } slide conn off() c := pb. NewMailClient (conn) /* Send */ MSG:= & pb. Message { From: Sender, To: [String{test@example.com}, Subject: Hello, Message_Text{ Message: Hello World, }, If _, err:= c.(context. err != nil { Fatalln log file (send:, err) } /* Get */ Stream, err:= c.Receive(context. Background(), & pb. Folder{}) If err != nil { Fatalln log (get:, err) } for { msg, err:= Recv(stream) if err==io EOF { break } if err != nil { log Fatalln (err:, err). It's not interesting because it's an example, but when you run a client with this server running, you're in a fixed message # Terminal A$ to run svr.go #terminal B$ to run cli.go in this article, I use simple RPC and RPC server-side streaming, but read the official guide for other formats such as RPC client-side streaming. In conclusion, I think it's a good idea to use it in the REST API if possible. Personally, I use the net/rpc package when using RPC, but I can't handle it from other languages with gRPC, we are available from popular languages and streaming, so we think it's easy to use. It's also not a direct advantage, but it's good not to have a cumbersome modeling because it automatically generates modeling code. Fenrir's developer account includes the latest developments for Fenrir products, follow us if you like! Follow related @fenrir_dev Follow related @fenrir_dev